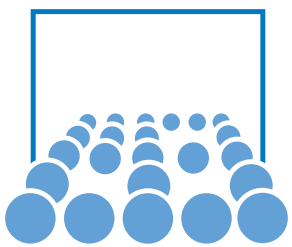
Fundamental Algorithms

Chapter 4: Selecting

Dirk Pflüger

Winter 2010/11



The Selection Problem

Definition (Selection Problem)

Input: a set A of n (distinct) numbers, and a number $i \in \{1, \ldots, n\}$.
Output: the element $y \in A$ (or its index, resp.) that is larger than exactly $i - 1$ other elements of A.

An immediate solution:

- sort A; effort: $\Theta(n \log n)$
- return element A[i] of the sorted array

Question:

Is there an asymptotically faster algorithm?

Finding Minimum and Maximum: $\Theta(n)$

```
Minimum(A:Array[1..n]) : Element {
    min := A[1];
    for i from 2 to n do {
        if min > A[i] then min := A[i];
    }
    return min;
}

Maximum(A:Array[1..n]) : Element {
    max := A[1];
    for i from 2 to n do {
        if max < A[i] then max := A[i];
    }
    return max;
}
```

QuickSelect

Idea: modify QuickSort

- select pivot and partition array
- only one partition needs to be further processed

```
QuickSelect(A:Array[p..r], i:Integer) : Element {
    if p=r then return A[p];

    q := Partition(A);

    k := q–p+1; // elements in the first partition

    if i <= k
       then return QuickSelect(A[p..q], i)
       else return QuickSelect(A[q+1 .. r], i–k);
}
```

Complexity of QuickSelect/RandSelect

Best Case:

- if we are very lucky, we find the element in only two partitioning steps $\rightarrow O(n)$.

Worst Case:

- if we always pick the smallest/largest element as pivot: $\Omega(n^2)$ (not cheaper than QuickSort)

"Intended" Case:

- half partition size in each step
- $T_{QS}(n) = n + T_{QS}(n/2)$; leads to $T_{QS}(n) \in \Theta(n)$

QuickSelect with Random Pivot

```
RandSelect(A:Array[p..r], i:Integer) : Element {
    if p=r then return A[p];

    q := RandPartition(A);

    k := q–p+1; ! elements in the first partition

    if i <= k
        then return RandSelect(A[p..q], i)
        else return RandSelect(A[q+1 .. r], i–k);
}
```

$\Rightarrow O(n)$ in the average case

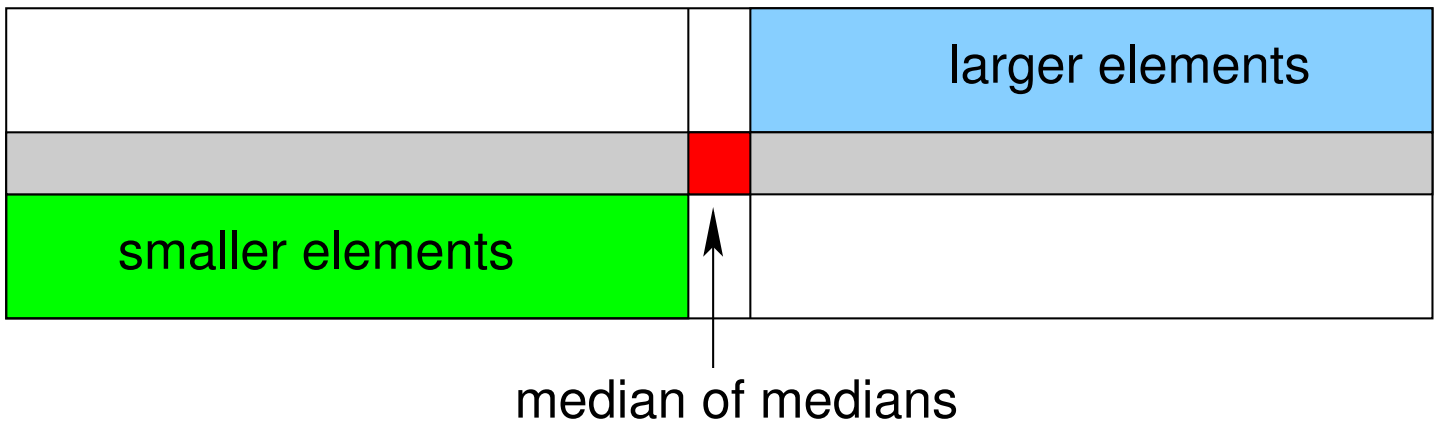
How to Guarantee a Good Pivot

Pivot element needs to be:

- larger/smaller than αn elements ($0 < \alpha < 1$)
- obtainable in $O(n)$ operations

The "Median of Medians" Idea:

- split array A into groups of five elements
- take median of each group
- use the median of these medians as pivot:



The BFPRT Algorithm

```
BFPRT_Select( A:Array[1..n], i:Integer) : Element {
    if n < C then return SortSelect(A,i);

    cols := ceil(n/5);  create Array M[1..cols];

    for i from 0 to cols-1 do {
        M[i+1] := Median5(A, 5*i+1, 5*i+5);
    }

    x := BFPRT_Select(M, floor(cols/2) );
    k := PivotPartition(A,x);

    if i <= k
    then return BFPRT_Select(A[1..k],i)
    else return BFPRT_Select(A[k+1 .. n],i-k);
}
```

The BFPRT Algorithm (2)

Ingredients:

- use a sorting-based algorithm ("SortSelect"), if n is small ($n < C$)
- Median5: algorithm to find the median of five elements:
  $\rightarrow$ can be derived via decision tree
- PivotPartition: partitioning according to a given value

Named after:

M. **B**lum, R.W. **F**loyd, V.R. **P**ratt, R.L. **R**ivest, and R.E. **T**arjan

The BFPRT Algorithm – Complexity

Size of the partitions:

- half of the “medians of five” are larger/smaller than the “median of medians” x
- half of the groups have 3 elements (incl. their median) that are smaller/larger than x:

$$3\left(\left\lceil\frac{1}{2}\cdot\left\lceil\frac{n}{5}\right\rceil\right\rceil-2\right)\geq\frac{3}{10}n-6$$

- worst-case partition size for the recursive call:

$$n-\left(\frac{3}{10}n-6\right)\leq\frac{7}{10}n+6$$

The BFPRT Algorithm – Complexity (2)

- recurrence equation:

$$T(n) \in \Theta(1) \qquad \text{for} \quad n < C$$

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7}{10}n + 6\right) + O(n) \qquad \text{for} \quad n \geq C$$

- assume: $T(n) \leq cn$ for some constant c

$$\begin{aligned} T(n) &\leq c\left\lceil \frac{n}{5} \right\rceil + c\left(\frac{7}{10}n + 6\right) + c_p n \\ &\leq c\left(\frac{n}{5} + 1\right) + \frac{7}{10}cn + 6c + c_p n = \frac{9}{10}cn + 7c + c_p n \\ &\leq cn, \qquad \text{if } \frac{1}{10}cn > 7c + c_p n \qquad \text{(i.e, choose } c\text{)} \end{aligned}$$

The BFPRT Algorithm – Complexity (3)

Result:

- BFPRT_Select requires $T(n) \in \Theta(n)$ operations

Corollary:

- use BFPRT_Select to find median pivot for QuickSort
  $\Rightarrow O(n \log n)$ worst-case complexity for QuickSort
- not used in practice (too large constants/effort for BFPRT_Select)